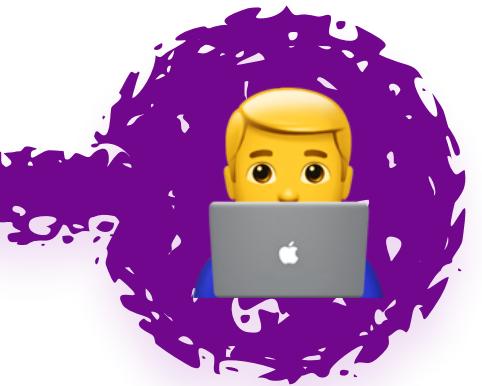


# Aula 02

Revisão da sintaxe, Exercícios e funções.

**Academy**



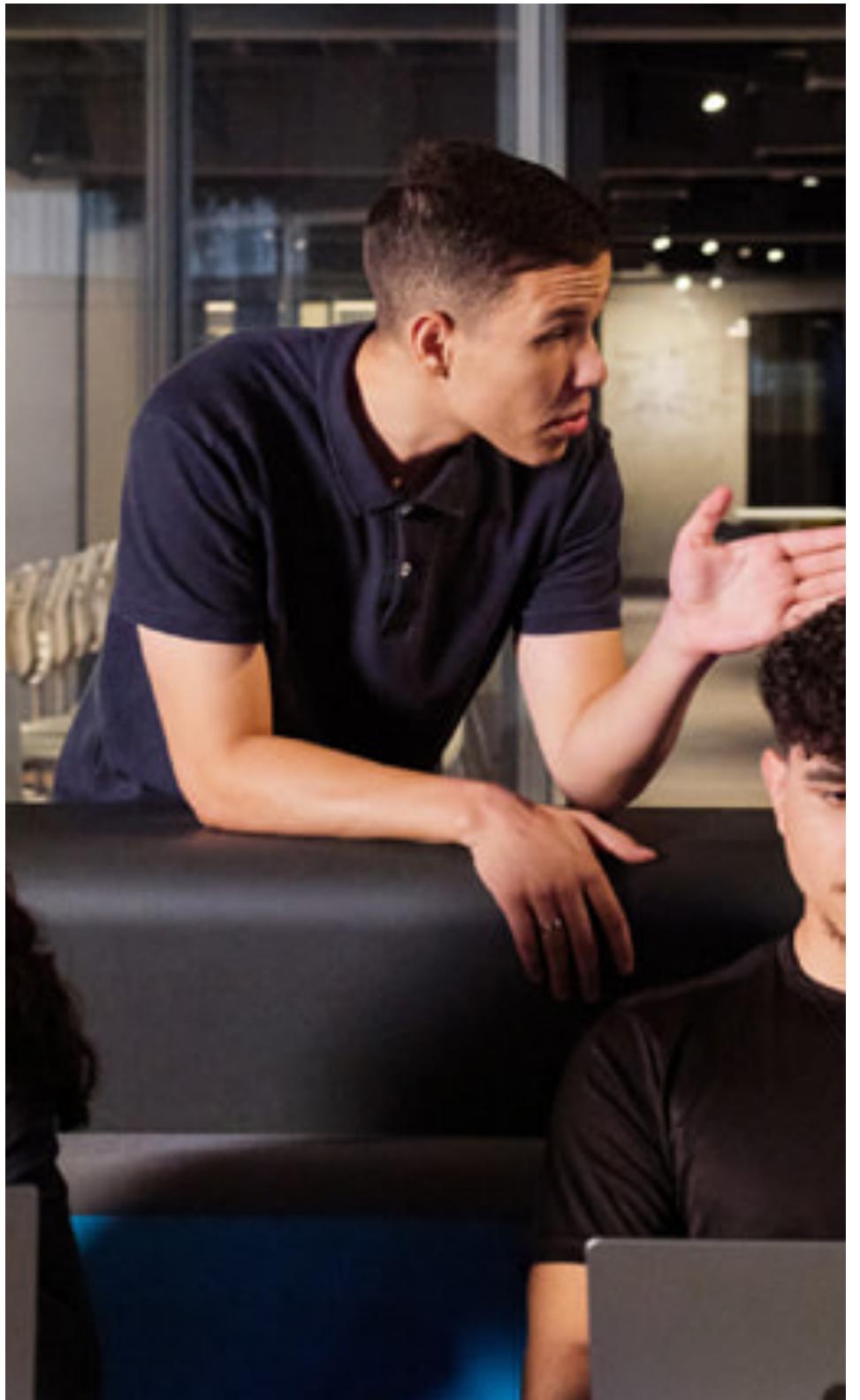
🧐 - Quem somos



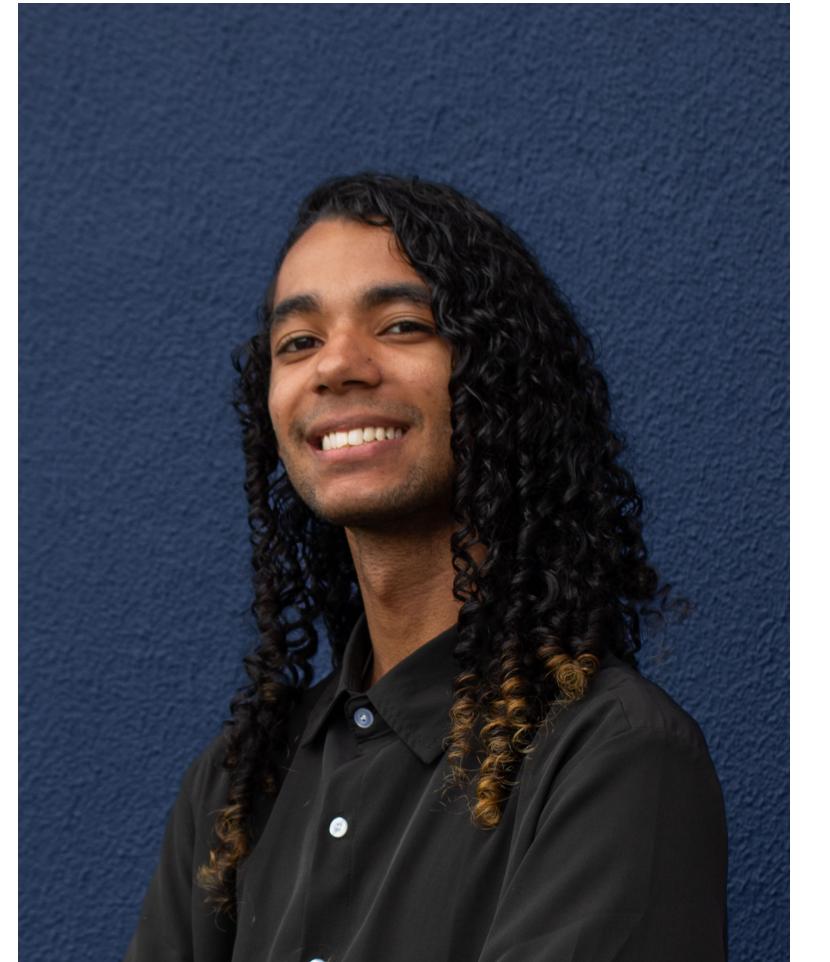
Leonardo Mesquita



Luca Gabriel



Kauã Miguel



João Victor

# **Sintaxe em C, Ponteiros, Alocação de Memória, Estrutura de dados, Programação Orientada a Objetos, Lógica Matemática**

**Nossa Prova - Conteúdos**

# Ponteiros

## Definição

- Uma variável que guarda um endereço de memória.
- Pode conter um endereço de memória OU null.
- O nulo é apenas uma representação de um endereço de memória inválido.

# Ponteiros

## Exemplo

```
#include <stdio.h>

int main(void) {
    int idade = 20;
    int *ponteiroParaIdade = &idade;
}
```

# Ponteiros

## Exemplo

```
int main(void) {
    int idade = 20;
    int *ponteiroParaIdade = &idade;

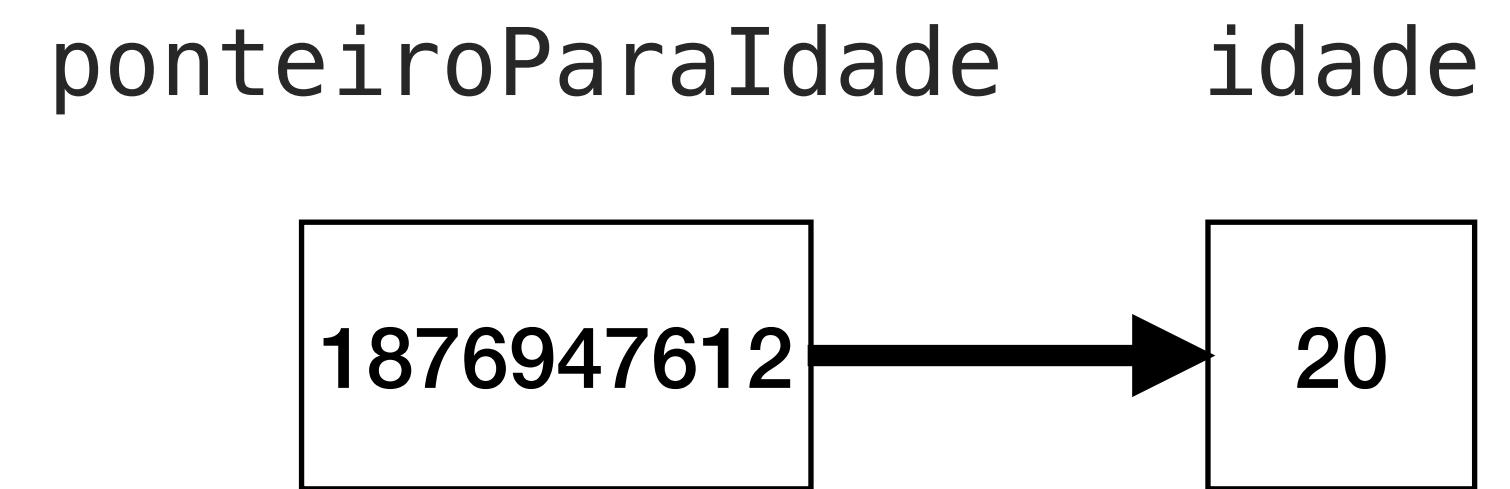
    printf("%d\n", idade); // Imprime 20
    printf("%d\n", &idade); // Imprime 1876947612
    printf("%d\n", ponteiroParaIdade); // Imprime 1876947612
}
```

# Ponteiros

## Exemplo

```
int main(void) {
    int idade = 20;
    int *ponteiroParaIdade = &idade;

    printf("%d\n", idade); // Imprime 20
    printf("%d\n", &idade); // Imprime 1876947612
    printf("%d\n", ponteiroParaIdade); // Imprime 1876947612
}
```



# Ponteiros

## Desreferenciação

- O operador \* nos permite acessar o valor para o qual o ponteiro aponta

# Ponteiros

## Exemplo

```
int main(void) {
    int idade = 20;
    int *ponteiroParaIdade = &idade;

    printf("%d\n", *ponteiroParaIdade); // Imprime 20
}
```

# Ponteiros

## Definição

- Precisamos definir o tipo de dado que o ponteiro aponta para fazer a aritmética de ponteiro.
- Essa expressão pula 4 byte para frente

```
int main (){  
    int x = 2;  
    int *p = &x;  
    p + 1  
}
```

# Ponteiros

## Definição

- Essa expressão pula 1 byte para frente

```
int main (){  
    char x = 'c';  
    char *p = &x;  
    p + 1  
}
```

# Ponteiros

## Qual o problema?

```
int main (){  
    int x,*p;  
    x=13;  
    *p=x;  
}
```

# Ponteiros

## Qual o problema?

```
int main (){  
    int x,*p;  
    x=13;  
    *p=x; // O ponteiro P não foi inicializado  
    // Estamos atribuindo 13 a um endereço desconhecido  
}
```

# Ponteiros

## Conceito

- Atribuições são feitos por referência
- Alterações refletem na variável original

# Ponteiros

## Exemplo

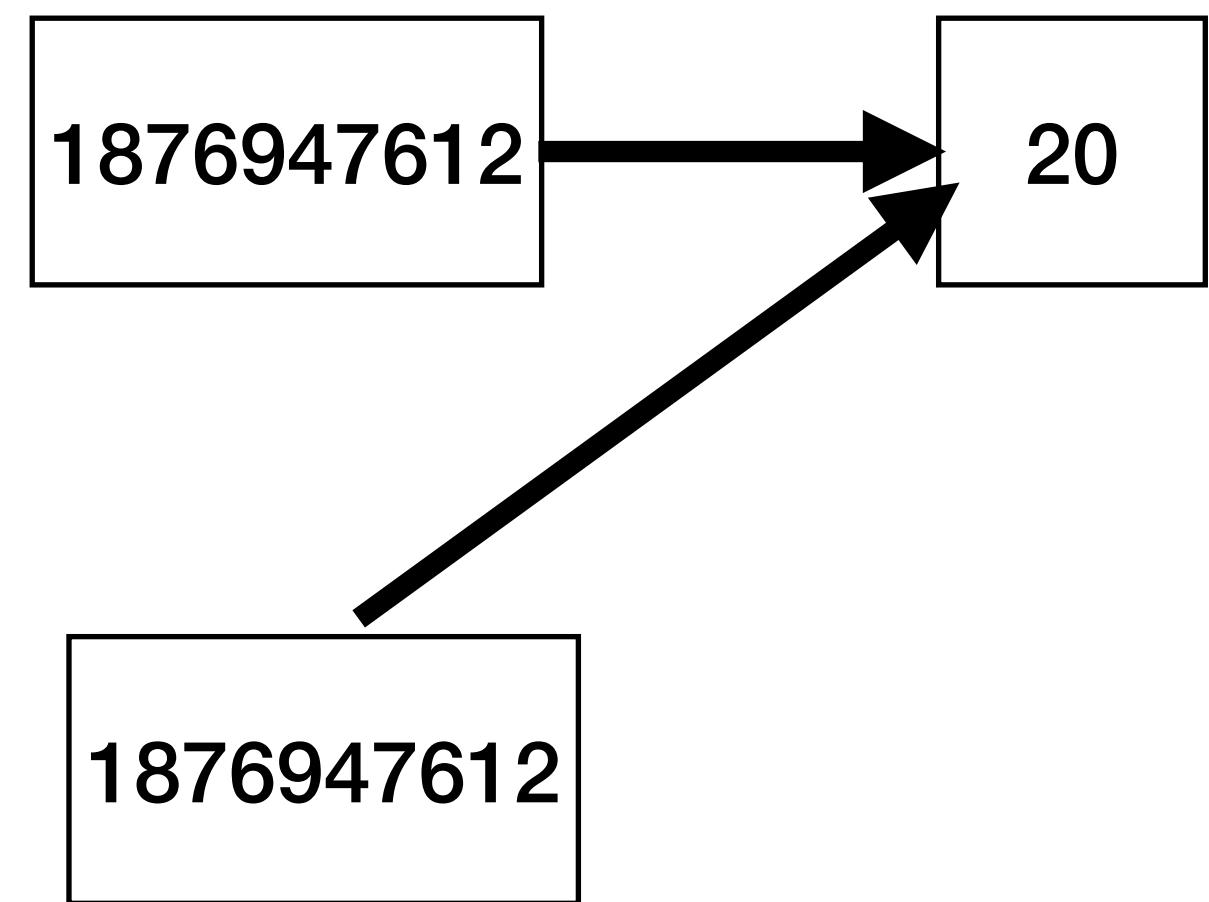
```
int main(void) {  
    int idade = 20;  
    int *ponteiroParaIdade = &idade;  
    int *segundoPonteiroParaIdade = &idade;  
}
```

# Ponteiros

## Exemplo

```
int main(void) {
    int idade = 20;
    int *ponteiroParaIdade = &idade;
    int *segundoPonteiroParaIdade = &idade;
}
```

ponteiroParaIdade      idade



segundoPonteiroParaIdade

# Ponteiros

## Exemplo

```
void atribuirValor(int *enderecoVariavel){  
    *(enderecoVariavel) = 21;  
}  
  
int main(void) {  
    int idade = 20;  
    atribuirValor(&idade);  
    printf("%d", idade); // Imprime 21  
}
```

# Ponteiros

Qual o valor de x?

```
int main(void) {  
    int x = 1;  
    int *p = &x;  
    int *p2 = p;  
    *p = *p + 2;  
    *p2 = *p2 + 3;  
    x++;  
    *p2 = *p + x;  
    printf("%d\n", x);  
}
```

- A) 11
- B) 12
- C) 13
- D) 14

# Ponteiros

Qual o valor de x?

```
int main(void) {  
    int x = 1;  
    int *p = &x;  
    int *p2 = p;  
    *p = *p + 2;  
    *p2 = *p2 + 3;  
    x++;  
    *p2 = *p + x;  
    printf("%d\n", x);  
}
```

- A) 11
- B) 12
- C) 13
- D) 14 

# Ponteiros

Qual o valor de x?

```
int main(void) {
    int x = 1;
    int *p = &x; // Aponta pra x
    int *p2 = p; // Aponta pra x
    *p = *p + 2;
    *p2 = *p2 + 3;
    x++;
    *p2 = *p + x;
    printf("%d\n", x);
}
```

# Ponteiros

Qual o valor de x?

```
int main(void) {  
    int x = 1;  
    int *p = &x;  
    int *p2 = p;  
    *p = *p + 2; // x = x + 2 = 3  
    *p2 = *p2 + 3; // x = x + 3 = 6  
    x++;  
    *p2 = *p + x;  
    printf("%d\n", x);  
}
```

# Ponteiros

Qual o valor de x?

```
int main(void) {  
    int x = 1;  
    int *p = &x;  
    int *p2 = p;  
    *p = *p + 2;  
    *p2 = *p2 + 3;  
    x++; // x = x + 1 = 7  
    *p2 = *p + x; // x = x + 7 = 14  
    printf("%d\n", x);  
}
```

# Ponteiros

Qual o valor de c?

```
int main(void) {  
    int c = 12;  
    int *p1 = &c;  
    int *p2 = p1;  
    *p1 = *p1 / 3;  
    *p2 = *p2 + 10;  
    c -= 5;  
    *p2 = *p1 - c;  
    printf("%d\n", c);  
}
```

- A) 1
- B) 0
- C) 9
- D) 14

# Ponteiros

Qual o valor de c?

```
int main(void) {  
    int c = 12;  
    int *p1 = &c;  
    int *p2 = p1;  
    *p1 = *p1 / 3;  
    *p2 = *p2 + 10;  
    c -= 5;  
    *p2 = *p1 - c;  
    printf("%d\n", c);  
}
```

- A) 1
- B) 0 
- C) 9
- D) 14

# Ponteiros

Qual o valor de c?

```
int main(void) {
    int c = 12;
    int *p1 = &c; // P1 aponta pra c
    int *p2 = p1; // p2 aponta pra c
    *p1 = *p1 / 3;
    *p2 = *p2 + 10;
    c -= 5;
    *p2 = *p1 - c;
    printf("%d\n", c);
}
```

# Ponteiros

Qual o valor de c?

```
int main(void) {
    int c = 12;
    int *p1 = &c;
    int *p2 = p1;
    *p1 = *p1 / 3; // c = 12 / 3 = 4
    *p2 = *p2 + 10; // c = 4 + 10 = 14
    c -= 5;
    *p2 = *p1 - c;
    printf("%d\n", c);
}
```

# Ponteiros

Qual o valor de c?

```
int main(void) {
    int c = 12;
    int *p1 = &c;
    int *p2 = p1;
    *p1 = *p1 / 3;
    *p2 = *p2 + 10;
    c -= 5; // c = 14 - 5 = 9
    *p2 = *p1 - c; // c = 9 - 9 = 0
    printf("%d\n", c);
}
```

# Ponteiros

Qual o valor de x?

```
int main(void) {  
    int x = 1;  
    int *p = &x;  
  
    for(; x < 5; x++) {  
        if(*p % 2 != 0){  
            *p = (*p) * x;  
        }  
    }  
    printf("%d\n", x);  
}
```

- A) 9
- B) 10
- C) 20
- D) 21

# Ponteiros

Qual o valor de x?

```
int main(void) {  
    int x = 1;  
    int *p = &x;  
  
    for(; x < 5; x++) {  
        if(*p % 2 != 0){  
            *p = (*p) * x;  
        }  
    }  
    printf("%d\n", x);  
}
```

- A) 9
- B) 10 
- C) 20
- D) 21

# Ponteiros

Qual o valor de x?

```
int main(void) {
    int x = 1;
    int *p = &x; // Tudo que mudar no p irá refletir no X.

    for(; x < 5; x++){
        if(*p % 2 != 0){ // Só entra se for ímpar
            *p = (*p) * x;
        }
    }
    printf("%d\n", x);
}
```

# Ponteiros

Qual o valor de x?

```
int main(void) {
    int x = 1;
    int *p = &x;

    for(; x < 5; x++){
        if(*p % 2 != 0){ // 1 é ímpar? Sim
            *p = (*p) * x; // x = 1 * 1 = 1
        }
    }
    printf("%d\n", x);
}
```

# Ponteiros

Qual o valor de x?

```
int main(void) {
    int x = 1;
    int *p = &x;

    for(; x < 5; x++) {
        // x = 3
        if(*p % 2 != 0){ // 3 é ímpar? Sim
            *p = (*p) * x; // x = 3 * 3 = 9
        }
    }
    printf("%d\n", x);
}
```

# Ponteiros

Qual o valor de x?

```
int main(void) {
    int x = 1;
    int *p = &x;

    for(; x < 5; x++){ // Incrementa e compara
        // 10 < 5 = false
        if(*p % 2 != 0){
            *p = (*p) * x;
        }
    }
    printf("%d\n", x); // Printa 10
}
```

# Ponteiros

Qual o output?

```
int main (){  
    int *i, *q, y = 0;  
    i = &y;  
    *i = 2;  
    y = *i + 1;  
    *i += 1;  
    ++*i;  
    (*i)++;  
    q = i;  
    printf("%d %d\n", *q, y);  
}
```

- A) 9 10
- B) 6 6
- C) 5 5
- D) 5 6

# Ponteiros

Qual o output?

```
int main (){  
    int *i, *q, y = 0;  
    i = &y;  
    *i = 2;  
    y = *i + 1;  
    *i += 1;  
    ++*i;  
    (*i)++;  
    q = i;  
    printf("%d %d\n", *q, y);  
}
```

- A) 9 10
- B) 6 6 
- C) 5 5
- D) 5 6

# Ponteiros

## Qual o output?

```
int main (){  
    int *i, *q, y = 0;  
    i = &y; // i aponta pra y  
    *i = 2; // o valor de i recebe 2, então y = 2  
    y = *i + 1;  
    *i += 1;  
    ++*i;  
    (*i)++;  
    q = i;  
    printf("%d %d\n", *q, y);  
}
```

# Ponteiros

Qual o output?

```
int main (){  
    int *i, *q, y = 0;  
    i = &y;  
    *i = 2;  
    y = *i + 1; // y = 2 + 1  
    *i += 1; // y = 3 + 1  
    ++*i; // y = 4 + 1  
    (*i)++; // y = 5 + 1  
    q = i; // q aponta pra i  
    printf("%d %d\n", *q, y); // Imprime 6 e 6  
}
```

# Ponteiros de ponteiros

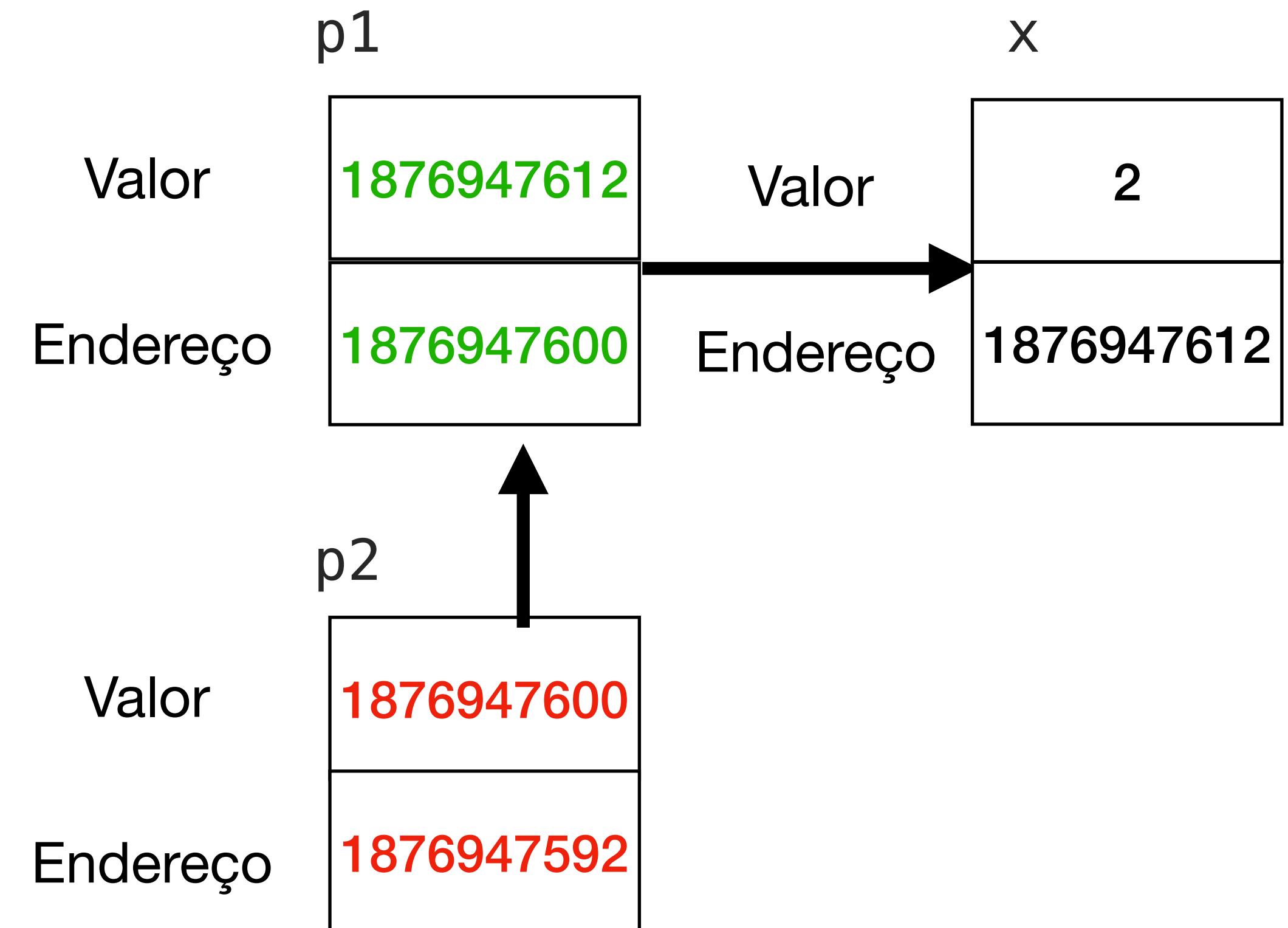
## Definição

- Uma variável que guarda um endereço de memória de outro ponteiro.
- Útil quando queremos lidar com matrizes, que são vetores de vetores.

# Ponteiros de ponteiros

## Exemplo

```
int main(void) {
    int x = 2, *p1, **p2;
    p1 = &x;
    p2 = &p1;
}
```



# Ponteiros de ponteiros

## Exemplo

```
int main(void) {
    int x = 2, *p1, **p2;
    p1 = &x;
    p2 = &p1;
    printf("%d\n", &p1); // Imprime 1876947600
    printf("%d\n", p2); // Imprime 1876947600
    printf("%d\n", &p2); // Imprime 1876947592
}
```

# Ponteiros de ponteiros

## Exemplo

```
int main(void) {
    int x = 2, *p1, **p2;
    p1 = &x;
    p2 = &p1;

    printf("%d\n", &x); // Imprime 1876947612
    printf("%d\n", *p1); // Imprime 2
    printf("%d\n", *p2); // Imprime 1876947612
    // *p2 o valor do endereço para qual p1 aponta
    // Ou seja, o endereço de x
}
```

# Ponteiros de ponteiros

Qual output?

```
int main (){
    int i=7, j=5, c;
    int *p;
    int **q;
    p = &i;
    q = &p;
    c = **q + j;
    printf("%d\n", c);
}
```

# Ponteiros de ponteiros

Qual output?

```
int main (){
    int i=7, j=5, c;
    int *p;
    int **q;
    p = &i; // p aponta pra variável i
    q = &p; // q aponta pro endereço do ponteiro p
    c = **q + j;
    printf("%d\n", c);
}
```

# Ponteiros de ponteiros

Qual output?

```
int main (){
    int i=7, j=5, c;
    int *p;
    int **q;
    p = &i;
    q = &p;
    c = **q + j; // **q é o valor do endereço que q aponta
    // q aponta pra p e p aponta pra i
    printf("%d\n", c);
}
```

# Ponteiros de ponteiros

Qual output?

```
int main (){
    int i=7, j=5, c;
    int *p;
    int **q;
    p = &i;
    q = &p;
    c = **q + j; // c = 7 + 5
    printf("%d\n", c);
}
```

# Ponteiros e struct

## Exemplo

```
typedef struct{
    int idade;
}Pessoa;

int main (){
    Pessoa kaua, *referencia;
    referencia = &kaua;
    printf("%d\n", &kaua.idade); // 1876947612
    printf("%d\n", referencia); // 1876947612
}
```

# Ponteiros e struct

## Exemplo

```
typedef struct{
    int idade;
}Pessoa;

int main (){
    Pessoa kaua, *referencia;
    referencia = &kaua;
    (*referencia).idade = 20;
    referencia->idade = 20;
    printf("%d\n", referencia->idade);
    printf("%d\n", (*referencia).idade);
}
```

# Ponteiros e struct

Qual valor de x?

```
typedef struct{
    char nome[20];
    char sobreNome[20];
}Pessoa;
```

```
int main (void){
    Pessoa joaquim, *referencia;
    char *p;
    int x;
    referencia = &joaquim;
    strcpy(referencia->nome, "joaquim");
    strcpy(referencia->sobreNome, "barbosa");
    p = strcat(referencia->nome, referencia->sobreNome);
    x = strlen(p);
    printf("%d", x++);
}
```

- A) 15
- B) 40
- C) 41
- D) 14

# Ponteiros e struct

Qual valor de x?

```
typedef struct{
    char nome[20];
    char sobreNome[20];
}Pessoa;
```

```
int main (void){
    Pessoa joaquim, *referencia;
    char *p;
    int x;
    referencia = &joaquim;
    strcpy(referencia->nome, "joaquim");
    strcpy(referencia->sobreNome, "barbosa");
    p = strcat(referencia->nome, referencia->sobreNome);
    x = strlen(p);
    printf("%d", x++);
}
```

- A) 15
- B) 40
- C) 41
- D) 14 

# Ponteiros e struct

Qual valor de x?

```
typedef struct{
    char nome[20];
    char sobreNome[20];
}Pessoa;

int main (void){
    Pessoa joaquim, *referencia;
    char *p;
    int x;
    referencia = &joaquim;
    strcpy(referencia->nome, "joaquim"); // Atribui joaquim a nome
    strcpy(referencia->sobreNome, "barbosa"); // barbosa no sobrenome
    p = strcat(referencia->nome, referencia->sobreNome);
    x = strlen(p);
    printf("%d", x++);
}
```

# Ponteiros e struct

Qual valor de x?

```
typedef struct{
    char nome[20];
    char sobreNome[20];
}Pessoa;

int main (void){
    Pessoa joaquim, *referencia;
    char *p;
    int x;
    referencia = &joaquim;
    strcpy(referencia->nome, "joaquim");
    strcpy(referencia->sobreNome, "barbosa");
    p = strcat(referencia->nome, referencia->sobreNome); // Concatena
    x = strlen(p);
    printf("%d", x++);
}
```

# Ponteiros e struct

Qual valor de x?

```
typedef struct{
    char nome[20];
    char sobreNome[20];
}Pessoa;

int main (void){
    Pessoa joaquim, *referencia;
    char *p;
    int x;
    referencia = &joaquim;
    strcpy(referencia->nome, "joaquim");
    strcpy(referencia->sobreNome, "barbosa");
    p = strcat(referencia->nome, referencia->sobreNome);
    x = strlen(p); // Pega o tamanho
    printf("%d", x++);
}
```

# Ponteiros e struct

Qual valor de x?

```
typedef struct{
    char nome[20];
    char sobreNome[20];
}Pessoa;

int main (void){
    Pessoa joaquim, *referencia;
    char *p;
    int x;
    referencia = &joaquim;
    strcpy(referencia->nome, "joaquim");
    strcpy(referencia->sobreNome, "barbosa");
    p = strcat(referencia->nome, referencia->sobreNome);
    x = strlen(p);
    printf("%d", x++); // Primeiro printa e depois atribui
}
```

# Alocação de memória

## Estática

- Um pedaço de memória reservado antes do programa executar, ou seja, em tempo de compilação.

# Alocação de memória

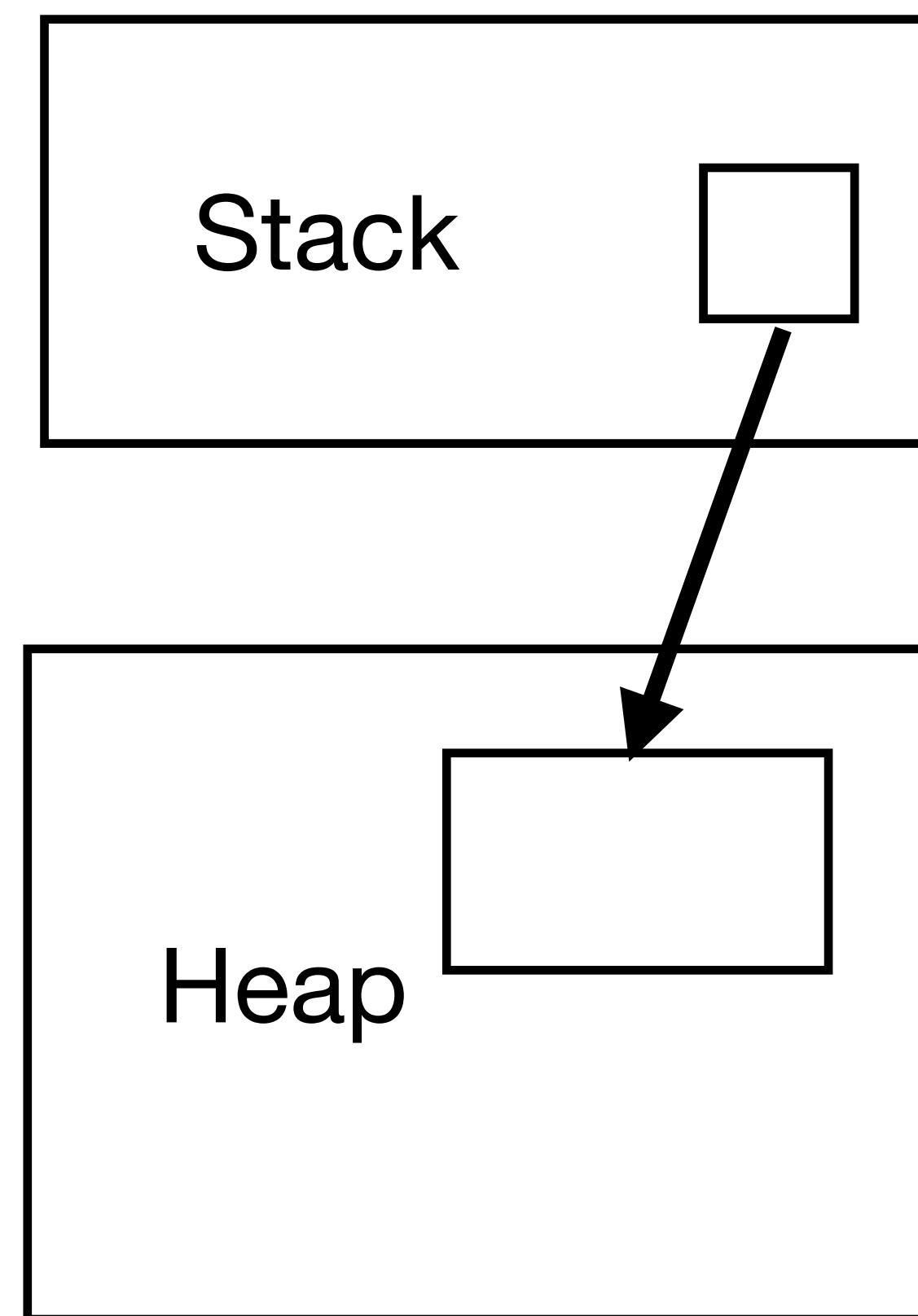
## Dinâmica

- Um pedaço de memória reservado enquanto o programa está em execução, ou seja, alocação de memória em tempo de execução.

# Alocação de memória

Como o C aloca memória para dados?

Memória virtual



# Alocação de memória

## Malloc

- Função que aloca memória no heap para guardar dados de tamanho dinâmico.

```
int main (void){  
    char frase[] = "Apple Developer Academy";  
    char *fraseHeap = malloc(sizeof(frase));  
}
```

# Alocação de memória

Qual o print?

```
int main (void){  
    char frase[] = "Apple Developer Academy";  
    char *fraseHeap = malloc(sizeof(frase));  
    printf("%s\n", fraseHeap);  
}
```

# Alocação de memória

Qual o print?

```
int main (void){  
    char frase[] = "Apple Developer Academy";  
    char *fraseHeap = malloc(sizeof(frase));  
    printf("%s\n", fraseHeap); // Nenhum, apenas alocou memória  
}
```

# Alocação de memória

Qual o print?

```
int main (void){  
    int *sequenciaValores = (int *)malloc(sizeof(int) * 3);  
    int *p = sequenciaValores;  
    *(p + 1) = 20;  
    *(sequenciaValores) = 40;  
    *(p + 2) = *(sequenciaValores) * 2;  
    printf("%d\n", *(p+2));  
}
```

# Alocação de memória

Qual o print?

```
int main (void){  
    //Aloca o tamanho 2 inteiros e retornar para a variável  
    int *sequenciaValores = (int *)malloc(sizeof(int) * 3);  
    int *p = sequenciaValores;  
    *(p + 1) = 20;  
    *(sequenciaValores) = 40;  
    *(p + 2) = *(sequenciaValores) * 2;  
    printf("%d\n", *(p+2));  
}
```

# Alocação de memória

Qual o print?

```
int main (void){  
    int *sequenciaValores = (int *)malloc(sizeof(int) * 3);  
    int *p = sequenciaValores; // Referencia p  
    *(p + 1) = 20;  
    *(sequenciaValores) = 40;  
    *(p + 2) = *(sequenciaValores) * 2;  
    printf("%d\n", *(p+2));  
}
```

# Alocação de memória

Qual o print?

```
int main (void){  
    int *sequenciaValores = (int *)malloc(sizeof(int) * 3);  
    int *p = sequenciaValores;  
    *(p + 1) = 20; // Atribui 20 ao primeiro endereço dinâmico  
    *(sequenciaValores) = 40;  
    *(p + 2) = *(sequenciaValores) * 2;  
    printf("%d\n", *(p + 2));  
}
```

# Alocação de memória

Qual o print?

```
int main (void){  
    int *sequenciaValores = (int *)malloc(sizeof(int) * 3);  
    int *p = sequenciaValores;  
    *(p + 1) = 20;  
    *(sequenciaValores) = 40; // Atribui 40 ao ponteiro seq  
    *(p + 2) = *(sequenciaValores) * 2;  
    printf("%d\n", *(p + 2));  
}
```

# Alocação de memória

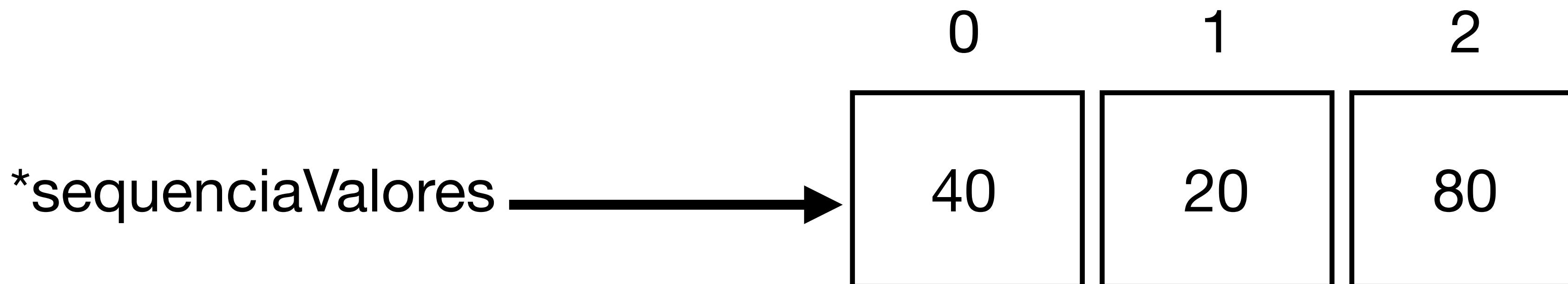
Qual o print?

```
int main (void){  
    int *sequenciaValores = (int *)malloc(sizeof(int) * 3);  
    int *p = sequenciaValores;  
    *(p + 1) = 20;  
    *(sequenciaValores) = 40;  
    *(p + 2) = *(sequenciaValores) * 2;  
    //Atribui o dobro do valor de seq no segundo endereço  
    //alocado  
    printf("%d\n", *(p + 2));  
}
```

# Alocação de memória

Qual o print?

```
int main (void){  
    int *sequenciaValores = (int *)malloc(sizeof(int) * 3);  
    int *p = sequenciaValores;  
    *(p + 1) = 20;  
    *(sequenciaValores) = 40;  
    *(p + 2) = *(sequenciaValores) * 2;  
    printf("%d\n", *(p + 2));  
}
```



# Alocação de memória

## Calloc

- Aloca um espaço na memória e inicializa os valores com 0.

```
int main (void){  
    int *sequenciaValores = calloc(3, sizeof(int));  
  
    printf("%d\n", *(sequenciaValores)); // Imprime 0  
    printf("%d\n", *(sequenciaValores + 1)); // Imprime 0  
    printf("%d\n", *(sequenciaValores + 2)); // Imprime 0  
}
```

# Alocação de memória

## Sizeof

- Calcula a quantidade em bytes para armazenar uma variável de um tipo de dado específico

```
int main(void) {  
    int x = 20;  
    printf("%d\n", sizeof(x)); // Retorna 4  
}
```

# Alocação de memória

Structs são consecutivas

```
typedef struct{
    int idade;
    float peso;
}Pessoa;

int main(void) {
    Pessoa kaua;
    printf("%d\n", &kaua.idade); // Imprime 1876947608
    printf("%d\n", &kaua.peso); // Imprime 1876947612
}
```

# Alocação de memória

Structs são consecutivas

```
typedef struct{
    int idade;
    float peso;
}Pessoa;

int main(void) {
    Pessoa kaua;
    printf("%d\n", &kaua.idade); // Imprime 1876947608
    printf("%d\n", &kaua.peso); // Imprime 1876947612
}
```

# Alocação de memória

Qual o resultado?

```
typedef struct{
    int idade;
    char nome[16];
}Pessoa;

int main(void) {
    Pessoa kaua;
    printf("%d\n", sizeof(kaua));
}
```

# Alocação de memória

Qual o resultado?

```
typedef struct{
    int idade;
    char nome[16];
}Pessoa;

int main(void) {
    Pessoa kaua;
    printf("%d\n", sizeof(kaua)); // Imprime 20
}
```

# Alocação de memória

## Free

- Libera memória que foi alocada.

# Alocação de memória

## Unions

- Um tipo de dado que agrupa valores de tipos diferentes em uma **mesma região de memória**.
- O que significa que o union armazena apenas um de seus membros em um momento.
- O tamanho é definido pelo maior membro do union

# Alocação de memória

## Unions

```
union Pessoa {  
    int idade;  
    double peso;  
    double altura;  
};  
  
int main(void) {  
    union Pessoa joaquim;  
    joaquim.altura = 1.8;  
    joaquim.peso = 79.0;  
    printf("%f\n", joaquim.altura); // Imprime 79.0  
}
```

# Alocação de memória

## Unions

```
union Pessoa {  
    int idade;  
    double peso;  
    double altura;  
};  
  
int main(void) {  
    union Pessoa joaquim;  
    printf("%d\n", sizeof(joaquim)); // Imprime 8  
}
```

**Obrigado!!!**



Kauã Miguel



Kauã Miguel



Kauamiguel\_